

AD-A051 745

INPUT OUTPUT COMPUTER SERVICES INC CAMBRIDGE MASS  
SINGLE-CHANNEL VOICE-RESPONSE-SYSTEM PROGRAM DOCUMENTATION. VOL--ETC(U)  
DEC 77

F/G 9/2

DOT-TSC-1107-2

UNCLASSIFIED

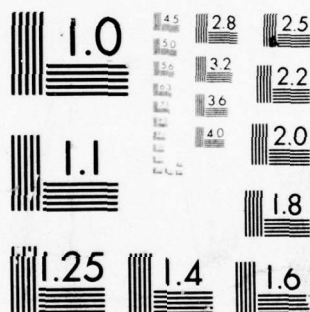
FAA/RD-77-177

NL

| OF |

AD  
A051 745





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



AD A051745

REPORT NO. FAA-RD-77-177, II

13

SINGLE-CHANNEL VOICE-RESPONSE-SYSTEM  
PROGRAM DOCUMENTATION  
Volume II: Program-Design Modules

Input Output Computer Services, Inc.  
689 Concord Avenue  
Cambridge MA 02138

AD No.  FILE COPY  




DECEMBER 1977

FINAL REPORT

DOCUMENT IS AVAILABLE TO THE U.S. PUBLIC  
THROUGH THE NATIONAL TECHNICAL  
INFORMATION SERVICE, SPRINGFIELD,  
VIRGINIA 22161

DDC  
RECEIVED  
MAR 24 1978  
B

Prepared for  
U.S. DEPARTMENT OF TRANSPORTATION  
FEDERAL AVIATION ADMINISTRATION  
Systems Research and Development Service  
Washington DC 20591

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

NOTICE

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.



(18) FAA/RD, TSC

(19) 77-177, FAA-77-24-2

Technical Report Documentation Page

1. Report No. FAA-RD-77-177, II ✓		2. Government Accession No.		3. Recipient's Catalog No. (12) T/P	
4. Title and Subtitle (6) SINGLE-CHANNEL VOICE-RESPONSE- SYSTEM PROGRAM DOCUMENTATION. Volume II. Program-Design Modules.		5. Report Date (11) Dec 1977		6. Report Date	
7. Author(s)		8. Performing Organization Report No. DOT-TSC-FAA-77-24, II ✓		9. Performing Organization Name and Address Input Output Computer Services, Inc.* 689 Concord Avenue Cambridge MA 02138	
10. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington DC 20591		11. Work Unit No. (TRAIS) FA83178109		12. Contract or Grant No. DOT-TSC-1107-2	
13. Supplementary Notes U.S. Department of Transportation ✓ *Under Contract to: Transportation Systems Center Kendall Square Cambridge MA 02142		14. Sponsoring Agency Code		15. Type of Report and Period Covered Final Report. Sep 1975 - Jan 1976.	
16. Abstract This report documents the design and implementation of a Voice Response System (VRS) using Adaptive Differential Pulse Code Modulation (ADPCM) voice coding. Implemented on a Digital Equipment Corporation PDP-11/20, this VRS system supports a single audio output channel. Vocabulary size is limited to 900 words or phrases. Input to the system consists of text messages or sentences in ASCII format transmitted to the 11/20 through a 300-baud asynchronous interface. A preliminary design for a VRS for 10 channels is reported.  This is the second of three volumes. Volume I is a system description, and Volume III is a user's guide.					
17. Key Words Voice Response System VRS ADPCM Speech Coding			18. Distribution Statement DOCUMENT IS AVAILABLE TO THE U.S. PUBLIC THROUGH THE NATIONAL TECHNICAL INFORMATION SERVICE, SPRINGFIELD, VIRGINIA 22161		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of Pages 68	
22. Price					

Form DOT F 1700.7 (8-72)

Reproduction of completed page authorized

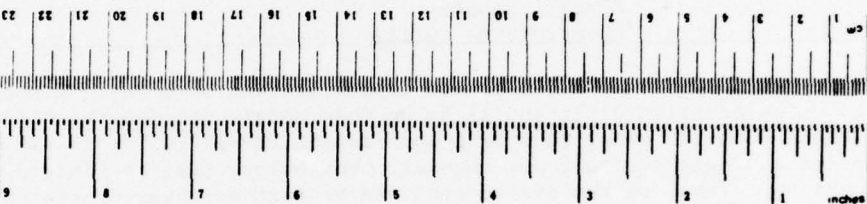
409553

Gu

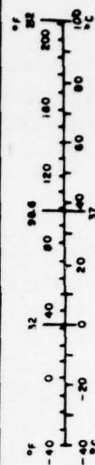
# METRIC CONVERSION FACTORS

## Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
<b>LENGTH</b>				
in	inches	2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
<b>AREA</b>				
sq in	square inches	6.5	square centimeters	cm <sup>2</sup>
sq ft	square feet	0.09	square meters	m <sup>2</sup>
sq yd	square yards	0.8	square meters	m <sup>2</sup>
sq mi	square miles	2.6	square kilometers	km <sup>2</sup>
acres	acres	0.4	hectares	ha
<b>MASS (weight)</b>				
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
<b>VOLUME</b>				
teaspoon	teaspoons	5	milliliters	ml
tablespoon	tablespoons	15	milliliters	ml
fluid oz	fluid ounces	30	milliliters	ml
cup	cup	0.24	liters	l
pint	pints	0.47	liters	l
quart	quarts	0.95	liters	l
gallon	gallons	3.8	liters	l
cubic foot	cubic feet	0.03	cubic meters	m <sup>3</sup>
cubic yard	cubic yards	0.76	cubic meters	m <sup>3</sup>
<b>TEMPERATURE (exact)</b>				
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C



Symbol	When You Know	Multiply by	To Find	Symbol
<b>LENGTH</b>				
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
km	kilometers	1.1	miles	mi
		0.6	miles	mi
<b>AREA</b>				
sq cm	square centimeters	0.16	square inches	in <sup>2</sup>
sq m	square meters	1.2	square yards	yd <sup>2</sup>
ha	hectares (10,000 m <sup>2</sup> )	0.4	square miles	mi <sup>2</sup>
		2.5	acres	ac
<b>MASS (weight)</b>				
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	ton
<b>VOLUME</b>				
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
		1.06	quarts	qt
m <sup>3</sup>	cubic meters	0.26	gallons	gal
		35	cubic feet	cu ft
		1.3	cubic yards	cu yd
<b>TEMPERATURE (exact)</b>				
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



## PREFACE

### GUIDE TO THE PROGRAM DOCUMENTATION

This volume contains:

#### CALLING SEQUENCES

A brief description of the major file management and text buffering routine is provided. The description includes the required arguments for the subroutines, error conditions, and a list of subroutines called by the described routine.

#### FLOW CHARTS

A flow chart is provided for each of the user commands available in VEDIT and RECORD. These charts assume knowledge of system operation, as described in the user manual (Volume III). Since copious reference is made to the listings, they should also be consulted.

#### LISTINGS

The program listings are found elsewhere. A running commentary is provided in addition to a short description of each program module. An index for the listings is found in Section 3 of this volume. The first part of the index gives the program names followed by the name of all source modules required to assemble the program. The second part gives the source module name followed by the name of all subroutines in that module.

#### LINKING CONVENTION

The system linking programs can be found in the section describing the module STKBUFF. The conventions used are described as an aid to understanding the attached flow charts.

Subroutines in the VRS have two possible returns. The first is the normal return. Execution continues after the call as normally would be expected from a subroutine return. The second is called an "error" return. This return is specified by providing an address the program should continue at, should the error return be taken.

It is important to note that the error return can mean one of two things: an actual error may have occurred in the system, such as an attempt to write the disk with the write-lock on; or, an error return can result from a test which fails. For example, DCTBM is a routine to find the entry in the dictionary which best matches the input string. If no match occurs,

for	
NIIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

however, the error return is taken. This does not indicate a program error, but rather is used to indicate that the input string does not match any entry in the dictionary. In the case of making sure that a new entry does not already exist, the error return is actually the desirable return for DCTBM.

The program flow charts indicate the error return by an arrow leaving the subroutine call, which is labelled "error". Again, this always indicates the error return as described above. Of course, the error return may be the desired return, thus, the reader should consult the listing or the calling sequence description to determine the conditions which lead to the error return.



## CONTENTS

<u>Section</u>	<u>Page</u>
1. INTRODUCTION	1
1.1 Hardware Environment	1
1.2 Software Environment	1
1.2.1 Vedit	1
1.2.2 Record	1
1.2.3 Speak	2
1.2.4 System Subroutines	2
1.2.5 Data Base	2
2. SOFTWARE	3
2.1 Vedit	3
2.2 Record	15
2.3 Speak	19
Phrase Look-Ahead Algorithm	21
3. SYSTEM SUBROUTINES	26
3.1 Program Assembly module names	27
3.2 Subroutine Names	28
3.3 Subroutine Description	35
4. DATA BASE	59
File System Description	

## ILLUSTRATIONS

<u>Figure</u>	<u>Page</u>
VEDIT - Main Program	5
Vedit Subroutines	
LIST	6
TALK	7
PRINT	7
INSERT	8
SN	9
SYNON	10
ENTER	11
RENAME	12
DELETE	13
Record Subroutines	
LISTEN	16
SAVE	17
SPEAK	20
3-1 POINTER SUMMARY ARROWS INDICATED DIRECTION OF MOTION	38
4-1 SAMPLE FILE STRUCTURE	60
4-2 FILE SYSTEM PARTITION	62

## TABLES

<u>Table</u>	<u>Page</u>
2-1 Sample Dictionary	23

## 1. INTRODUCTION

This documentation describes the operation of the programs comprising the single channel Voice Response System (VRS) delivered under contract DOT/TSC 1107. The descriptions include a short narrative of each module and corresponding flow charts. The program listings are furnished under separate cover.

### 1.1 HARDWARE ENVIRONMENT

The single channel VRS system requires a PDP-11/20 computer configured as shown in Figure 1. At least 12K core memory is required.

### 1.2 SOFTWARE ENVIRONMENT

The programs described run under control of the DEC RT-11 operating system, version 2. The single job (SJ) monitor is used. There are three main programs callable from the monitor level as follows. Each program is treated in detail in subsequent sections.

#### 1.2.1 VEDIT

Program VEDIT comprises all modules required to create, modify and update the dictionary which maps ASCII names to the disk resident voice files. VEDIT contains a command string interpreter (CSI) for reacting to user input. A complete description of VEDIT and other program commands is given in the user manual.

#### 1.2.2 RECORD

Program RECORD comprises all modules required to enter audio speech utterances into the system. It contains modules which: (1) accept audio input and digitize it into a temporary file; (2) process the file by the ADPCM algorithm; (3) auto-edit leading and trailing silence; and (4) associate each utterance with a dictionary entry and build the disk-based voice file.



#### 1.2.3 SPEAK

Program H516 comprises all modules required to generate audio from the voice files. The modules accept ASCII text from the H516 computer, search for the disk blocks containing the voiced text, and decode the voice file into the audio signal which currently drives a speaker. H516 contains the routines for parsing the input text, including insertion of pauses, identification of the best text match using phrase look-ahead, and proper interpretation of numbers.

#### 1.2.4 System Subroutines

All programs make extensive use of shared subroutines which perform specific tasks, such as buffer management, pattern matching, register saving, etc. All subroutine modules of general interest are documented fully, and those routines unlikely to be frequently required have a brief description of the routine as well as its calling parameters. All routines are identified in the program listings.

#### 1.2.5 Data Base

The dictionary and voice file are stored in a single disk file: "DIRECT.DVF". The generation and management of the data base is described below.

## 2. SOFTWARE

### 2.1 VEDIT

VEDIT is entered through the global symbol "CSI". Upon entry the program version number is printed and the system initialization routine is called. The system initialization routine opens file DIRECT.DVF if it exists or creates a new one if it does not. If an error occurs in system initialization, the program exits, since the program is unable to proceed without proper initialization. If no error occurs, the main loop is entered.

Command input occurs first in the main loop. A question mark is printed and type-in is accepted from the console terminal. The command input routine does not return until either a new character is entered or until an error occurs. The only error likely to occur is an attempt by the user to input a command which is more than 255 characters long.

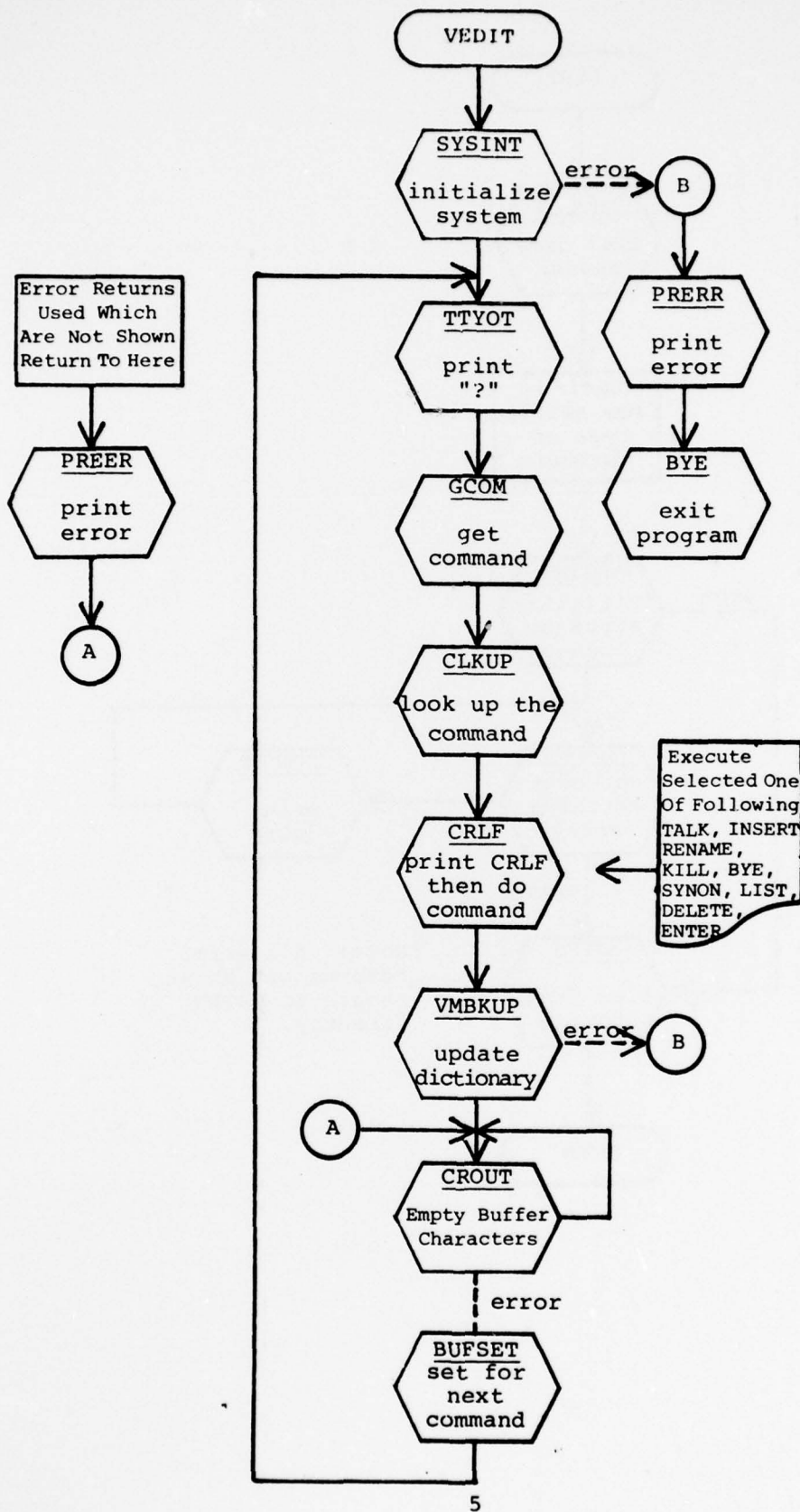
The command entered is now looked up in a table of commands. The command is matched, character for character, with command names in the table until the first break character. Only enough of the command name need be typed to prevent matching more than one command. When the command is matched, the command string is also checked for any switch options to the command. If a switch exists, the switch character is put in the global variable "SWITCH". Finally, a pointer to the matching routine is returned.

The routine selected by command lookup is executed. The remainder of the entered command is passed to the routine for use as an argument list. The command routine also has access to SWITCH and other globals in the program to insure its proper execution. Upon completion of the routine, the remainder of the command string not used is ignored and the disk copy of file DIRECT.DVF is updated if any changes to temporarily core resident sections have been made.

Errors occurring are of two types. The first can be called "recoverable". These are such things as user typographical errors or undefined or incorrect arguments. In this case a message is printed and a carriage return line feed is printed. The second are errors such as a failure of the initialization routine. Errors of this type indicate hardware or an RT-11 operating system error. The program itself is incapable of handling such errors and so the program exits.

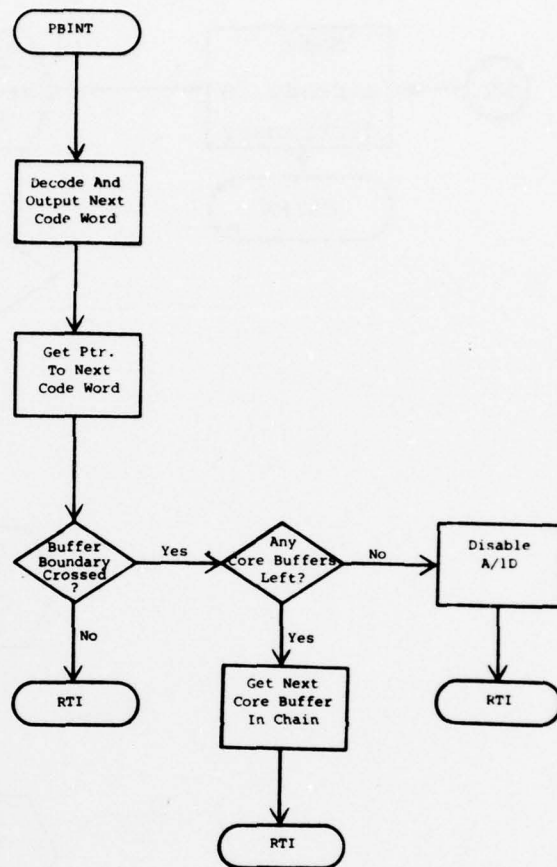
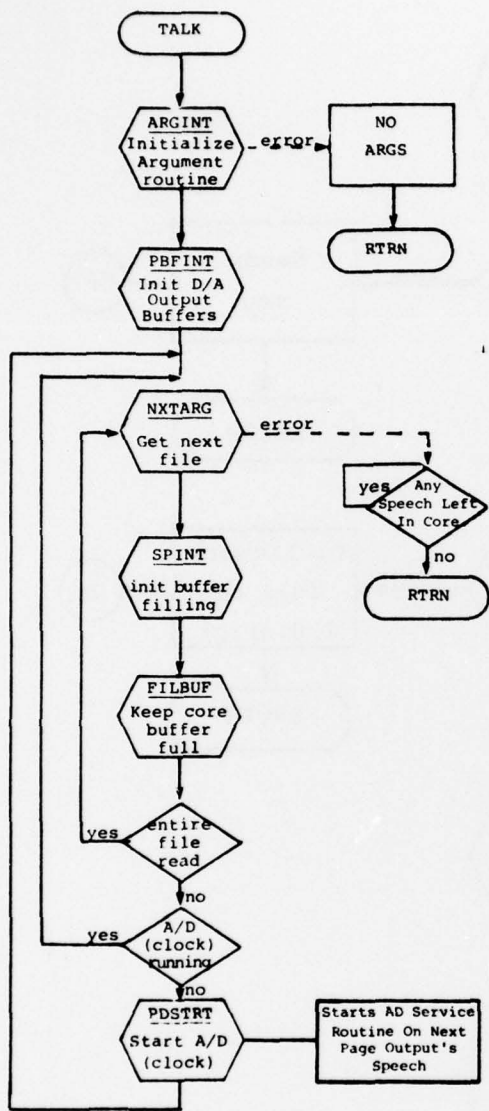
The following section contains flow charts of the basic commands callable from the VEDIT command string interpreter.

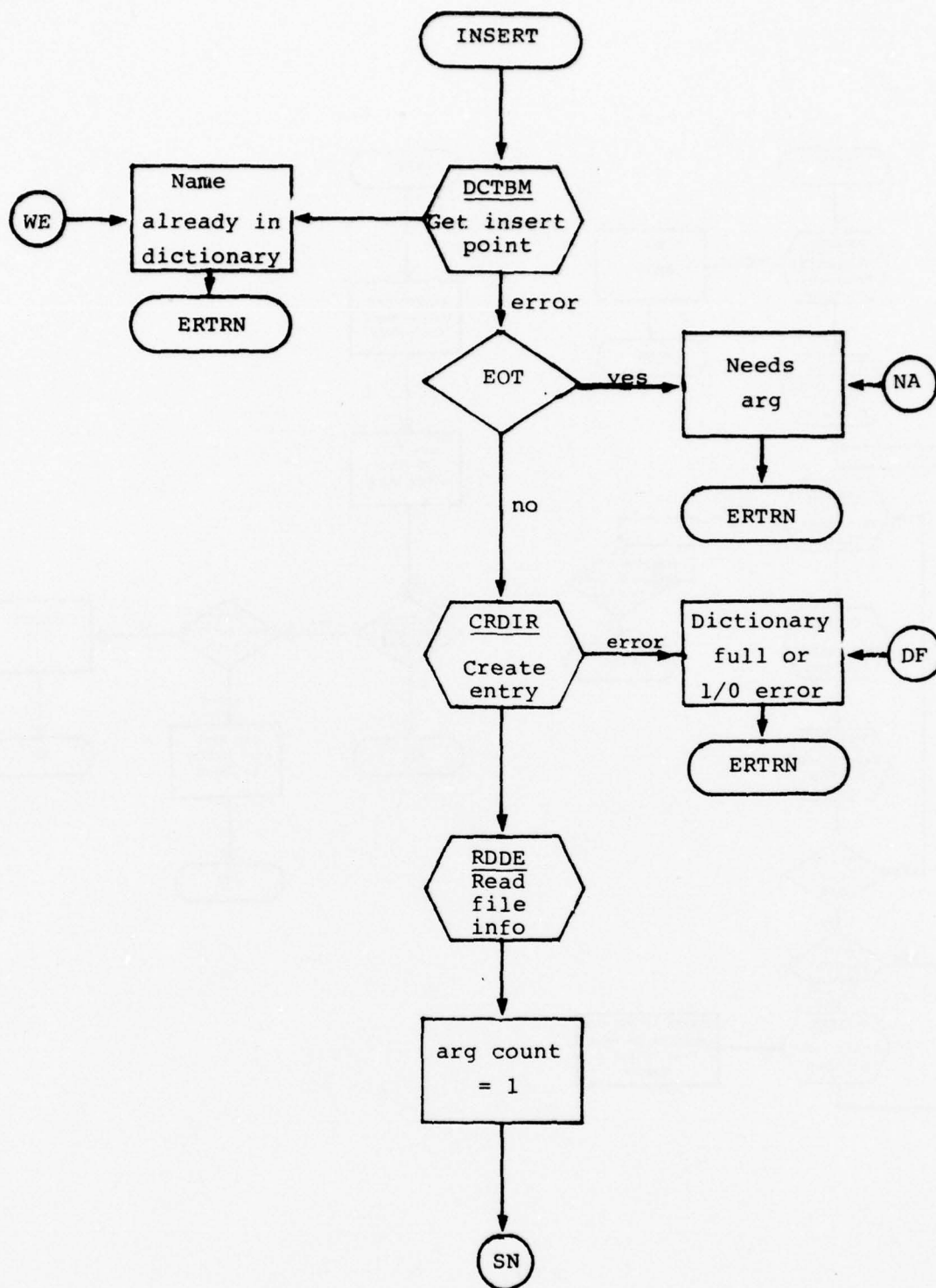
The description of the commands described in the VRS users manual provides guidelines for following the flow charts. In addition, there are descriptions of the major subroutines given at the end of this section. Consulting the program listings is also helpful.



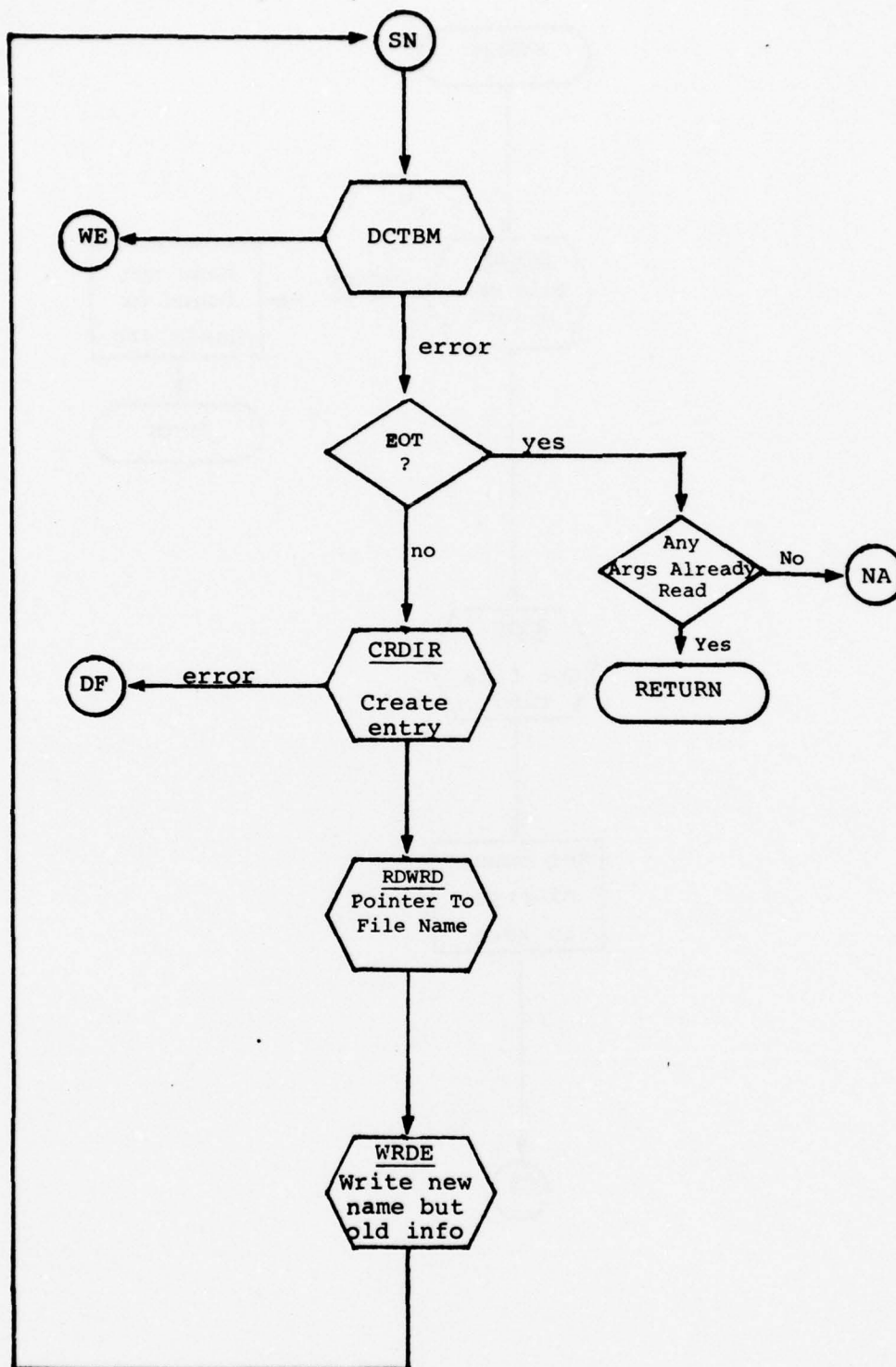


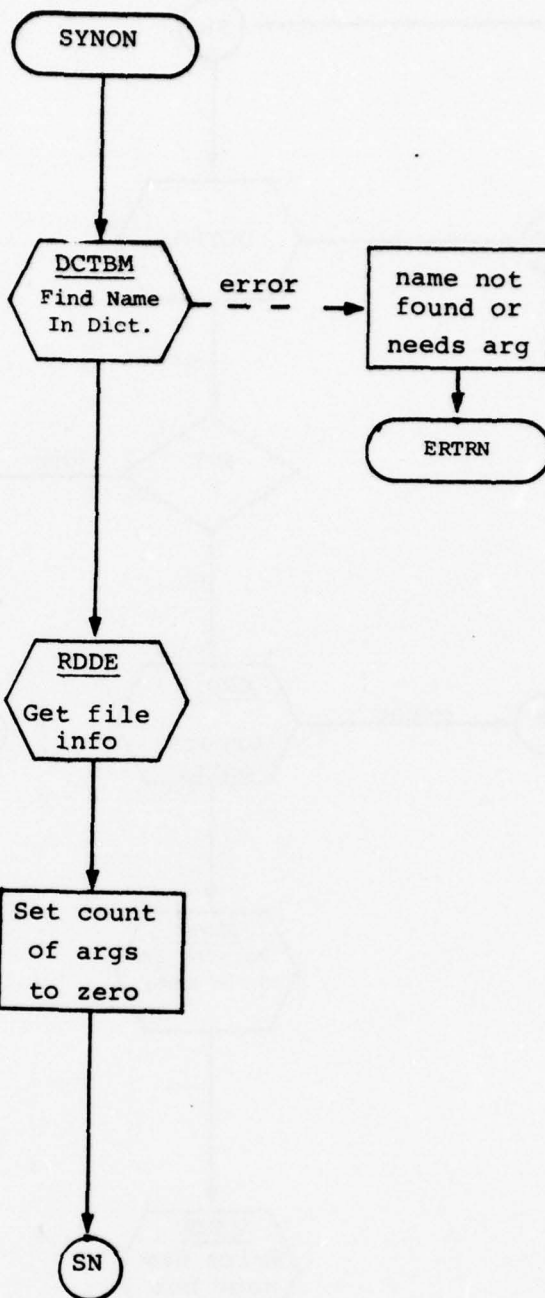


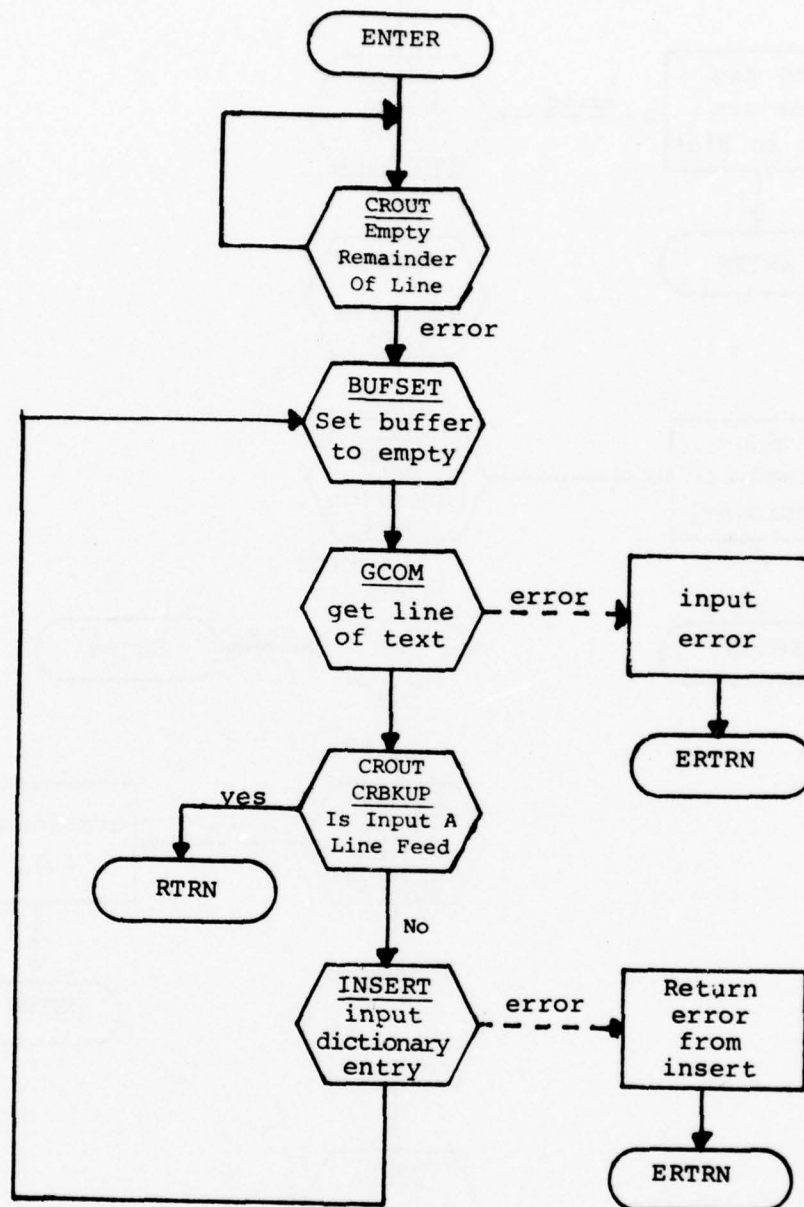


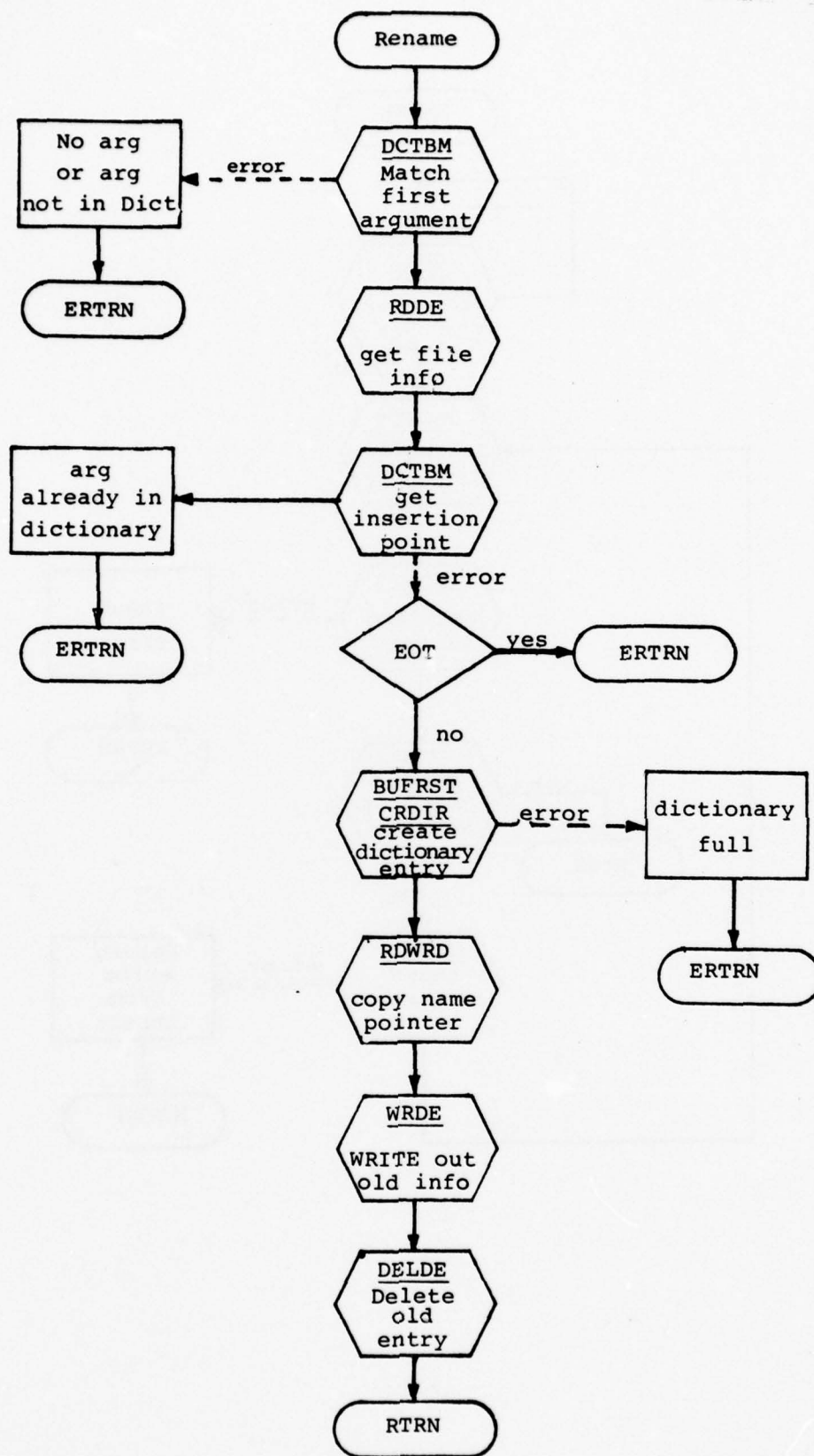


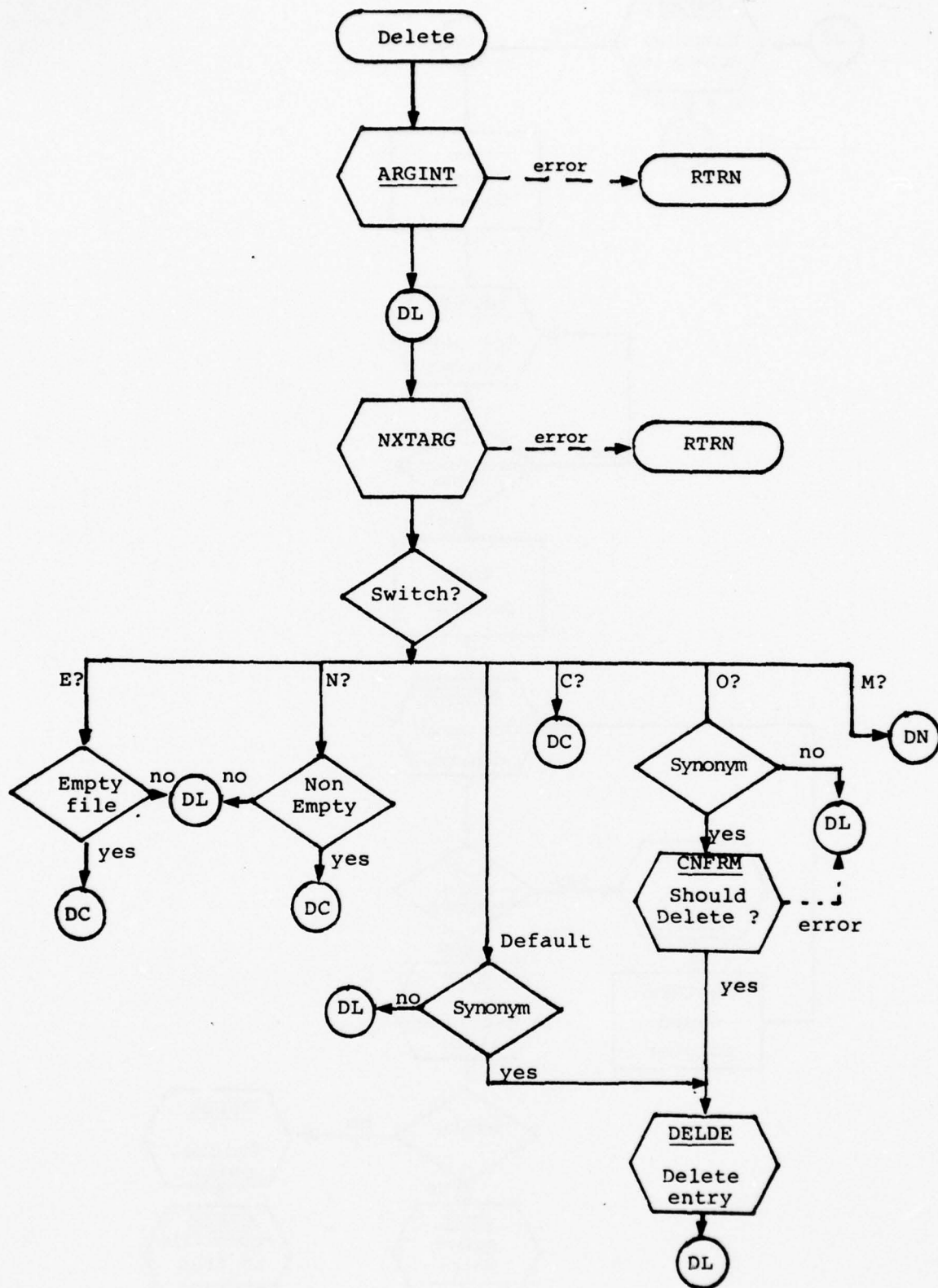




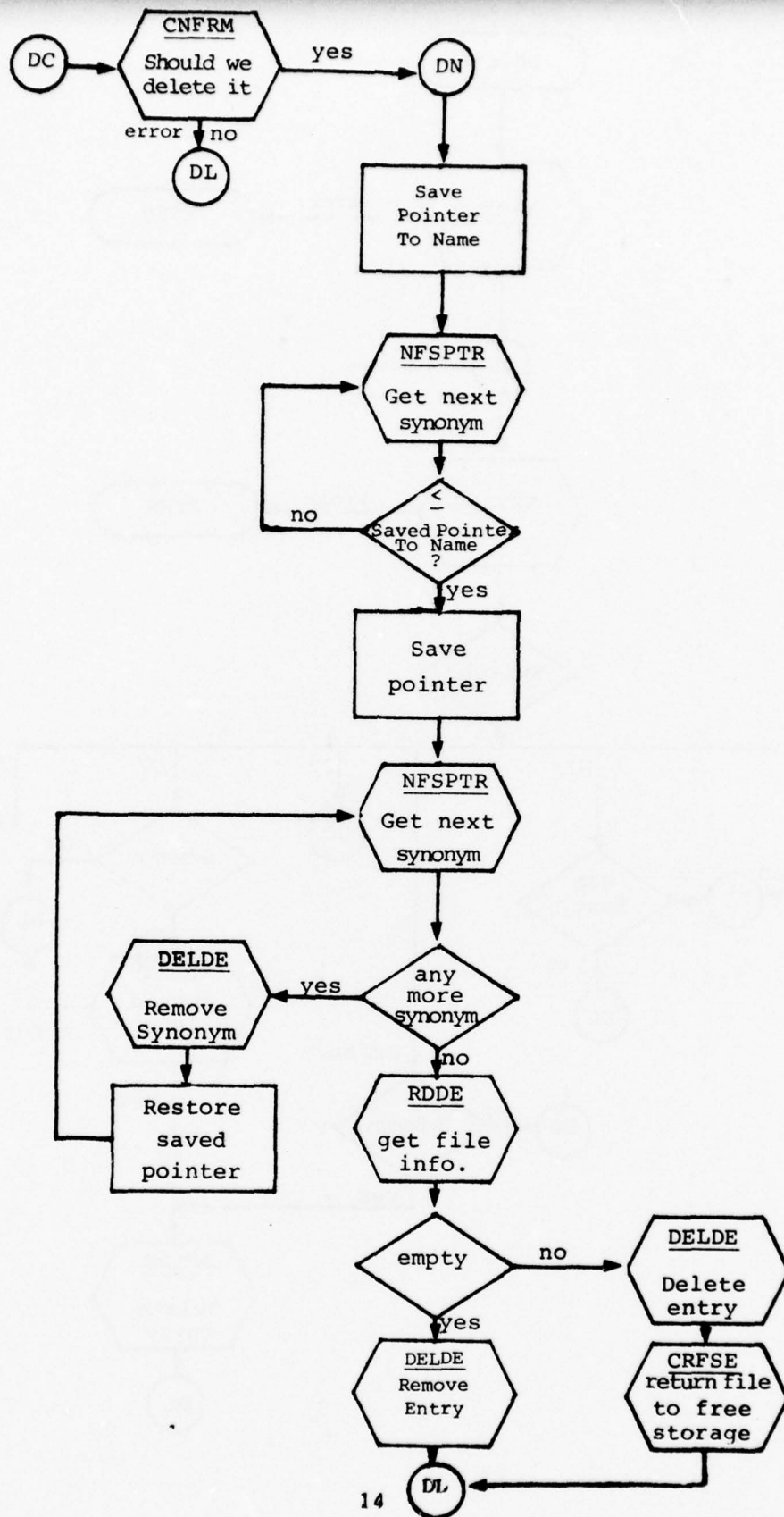












## 2.2 RECORD

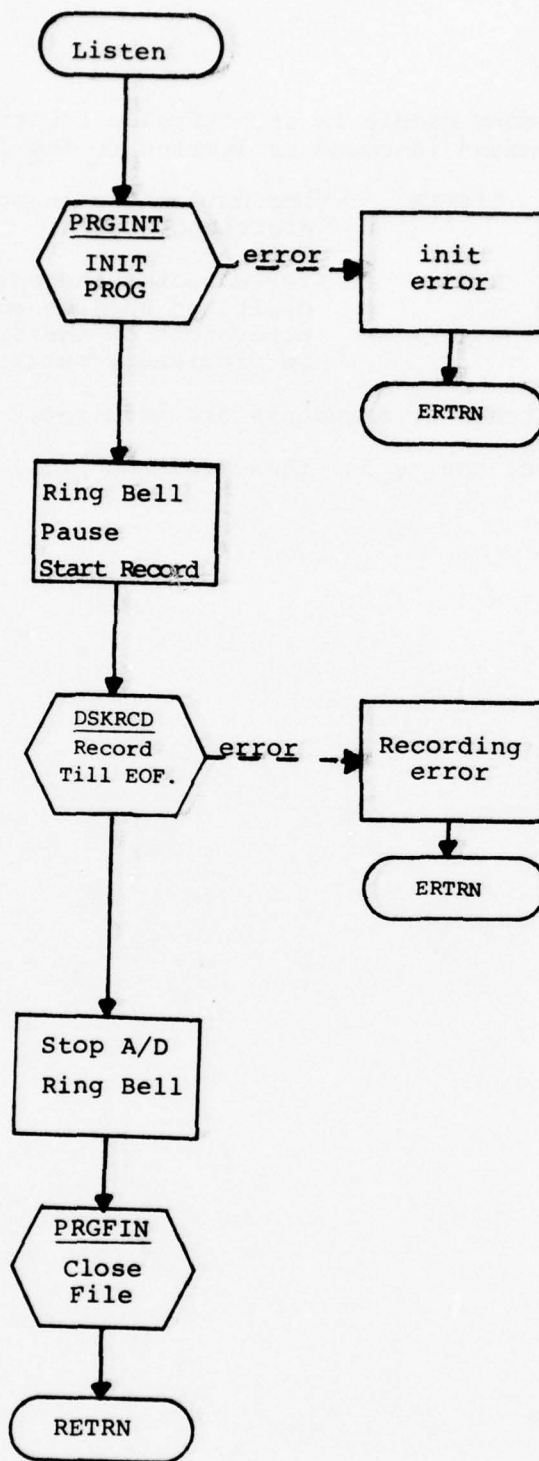
The record module is structurally identical to VEDIT.  
The command language is limited to the following:

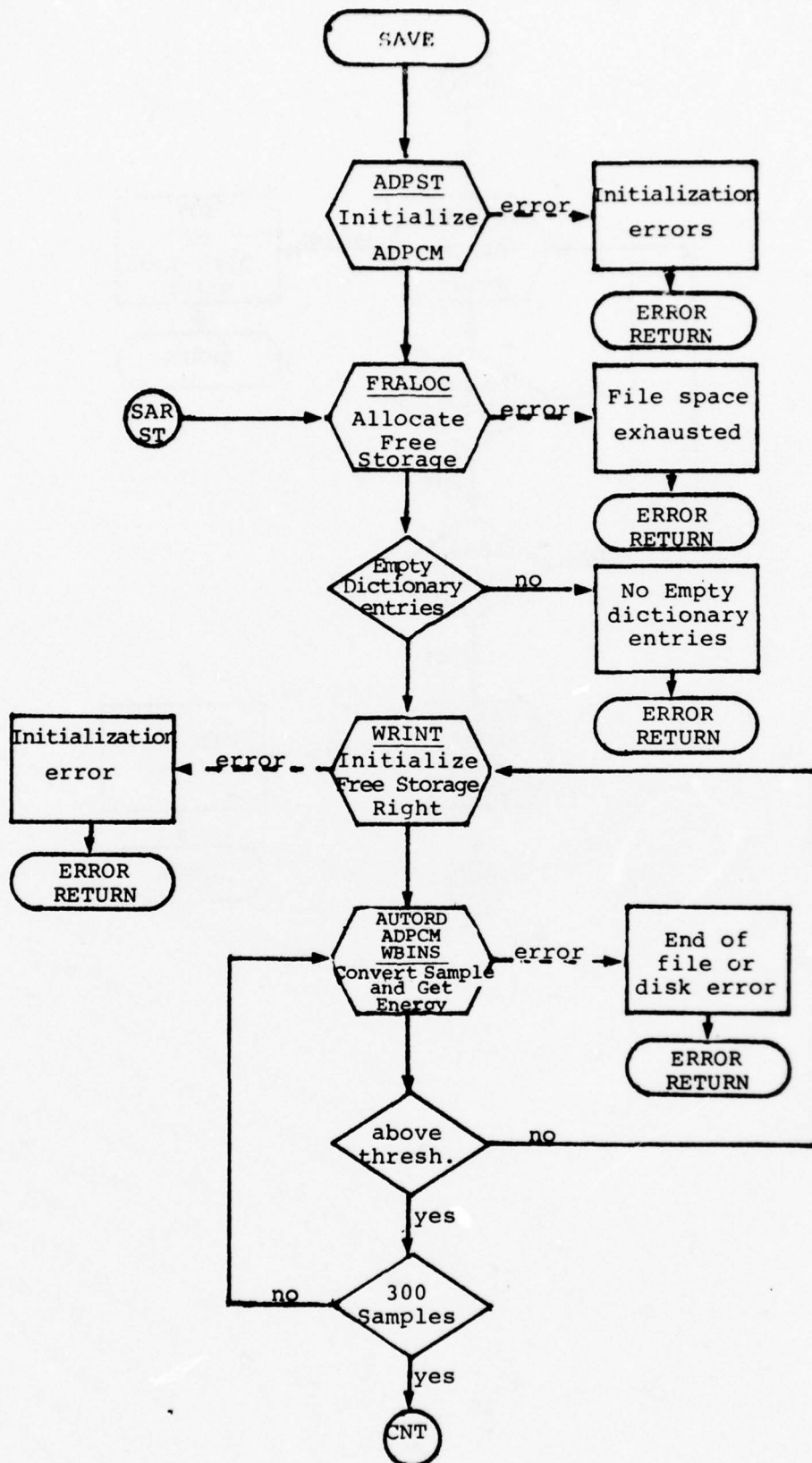
LISTEN	(for building a temporary file of digitized speech)
SAVE	(for encoding and editing the digitized speech, and cataloguing utterances on the disk according to dictionary entries)

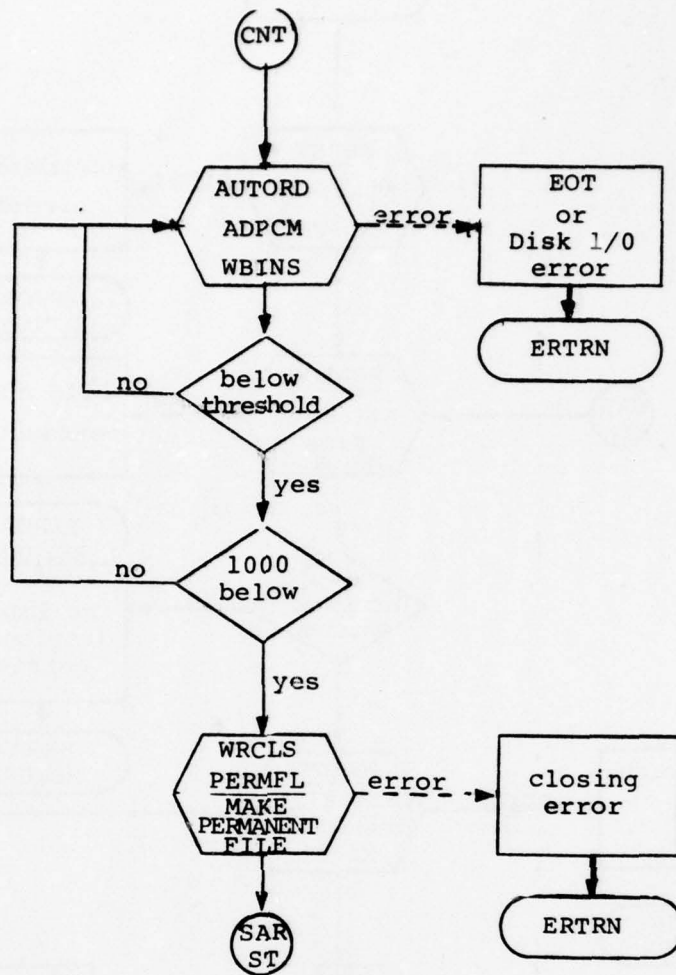
No switches or arguments are required.

The flow charts for these commands follows.





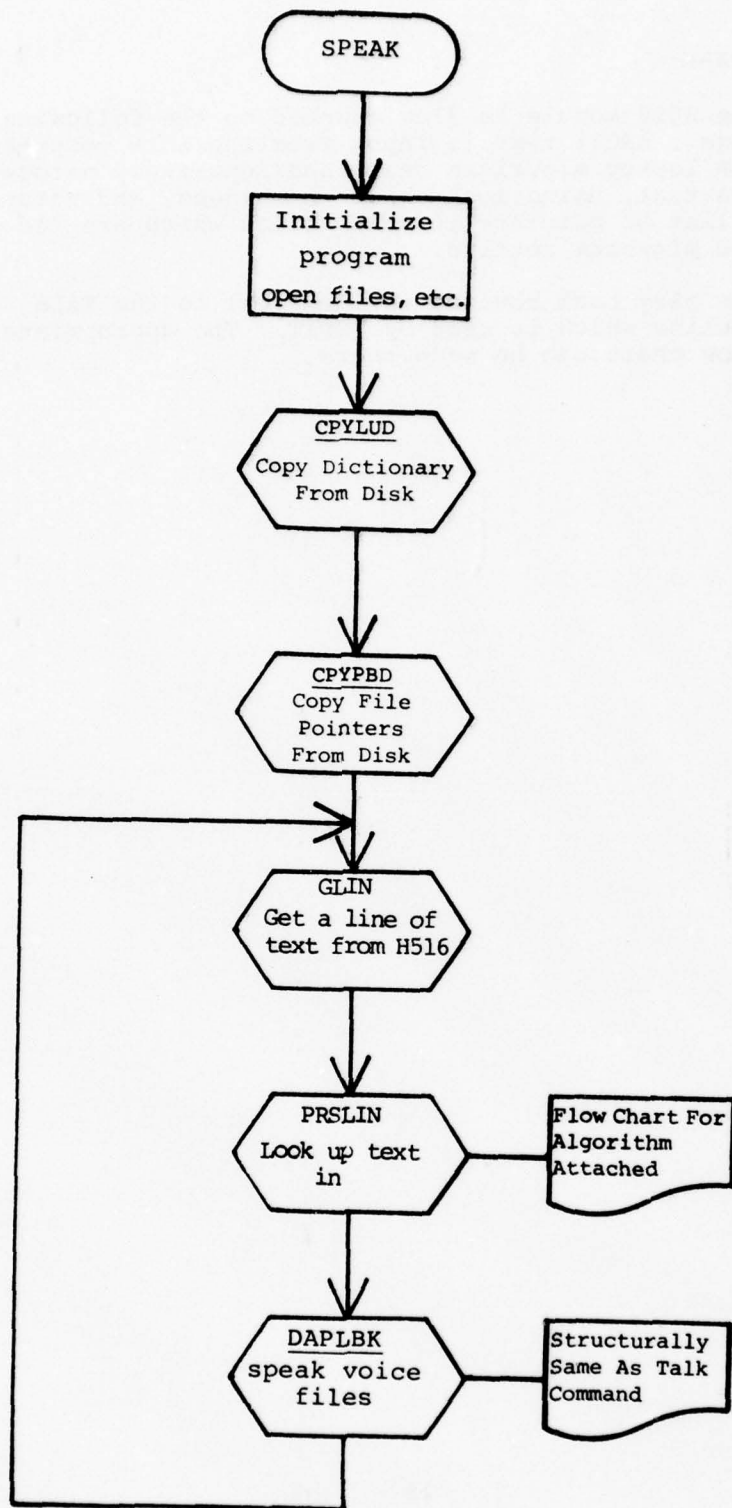




### 2.3 SPEAK

The H516 module is flow charted on the following page. ASCII text is input from the H516 computer. The lookup algorithm described separately parses the text, using look-ahead techniques, and returns a list of pointers to voice files which are fed to the playback routine.

The play back routine is identical to the TALK routine which is used by VEDIT. The appropriate flow chart can be seen there.





### Phrase Look-ahead Algorithm

This section is devoted to an analysis of the "phrase look-ahead" function of the H516 program. Several problem areas will be dealt with. A familiarity with a standard binary search is assumed.

Consider the dictionary listed in Table 2-1 and the text string "NEW YORK CITY IS A LARGE PLACE". There are two problems that must be dealt with. First, "NEW" matches two entries - "NEW" if taken alone or "NEW YORK CITY" if taken as part of a phrase. "NEW YORK CITY" is the better match, but if the binary search arrived at "NEW" first, a match would be detected and the search would terminate prematurely.

The solution lies in the fact that a phrase which matches a string will always be alphabetically later in the dictionary than a word or shorter phrase which matches the same string. Therefore, whenever a match occurs before the end of the search, (before  $\log_2 N$  tries for a dictionary of  $N$  entries), the matching entry is stored. The match is then treated as a mismatch which is less than the string input. At the end of the search the most recently encountered match will be the best match.

The second problem is somewhat more subtle. Consider again the dictionary in Table 2-1, and the input string "NEW YORK STATE". The first try with the binary search would compare the input string with "LASTING" after which it would try "NEW JERSEY". The character where the mismatch occurs is at "Y" in the input and "J" in Jersey. This would indicate that the input is greater than the entry "NEW JERSEY" so the binary search would proceed to "NEWARK". A blank is alphabetically less than an "A", so the binary search would next try "NEW YORK CITY". This would be its last try. But the "S" in "STATE" mismatches the "C" in "CITY". Therefore, the binary search would indicate no match even though "NEW" by itself does match an entry.

The problem occurred at the entry, "NEW JERSEY". The comparison indicated that the input is greater than the dictionary entry because "Y" is greater than "J". But since this mismatch occurred after a blank was encountered, the matching entry will be greater than the input only if a phrase with at least one blank will be the final match. If the final match has no imbedded blanks, then the matching entry will always be less than the entry where the mismatch occurred after a blank was encountered.

The binary search will go in the wrong direction after a mismatch if several conditions occur. The first condition is that the number of blanks encountered before the mismatch occurs is greater than the number of blanks that will occur in the best possible match. The second is that the result of the mismatch indicates that the matching entry is greater than the entry just compared with the input string, as in comparing "NEW YORK" with "NEW JERSEY".

The solution involves a "tree search" of the dictionary whenever no blanks are encountered. When a mismatch is encountered the binary search proceeds as normal. If, however, one or more blanks are encountered in the mismatch and the mismatch directs the search to proceed in the "greater than" direction, the point in the search where this occurred is pushed onto the stack and then the search is permitted to continue in the "greater than" direction. At the end of the search done in that way, the location in the search which was pushed on the stack is popped off and the search proceeds again from that point but in the "less than" direction. As described in the first problem area, the searches are allowed to continue even if matches are encountered. The best match will be the match containing the most imbedded blanks.

This can occur any number of times, and several of these decision points can be on the stack at once. Also note that if one is proceeding from a point at which the mismatch contained one blank and is moving in the "greater than" direction, the further mismatches must contain two or more blanks before they can be saved on the stack. This simplifies the tree search somewhat and increases the speed of the search.

A flow chart of the algorithm aids in understanding this operation. It is helpful to draw a dictionary as a binary tree and use it to trace the search for various inputs to find the path followed.



TABLE 2-1. SAMPLE DICTIONARY

DENSE

DENSE AIR

DENSE FOG

DENSE SMOKE

LAST CHANCE

LAST ENTRY

LAST FLIGHT

LASTING

M

N

NEW

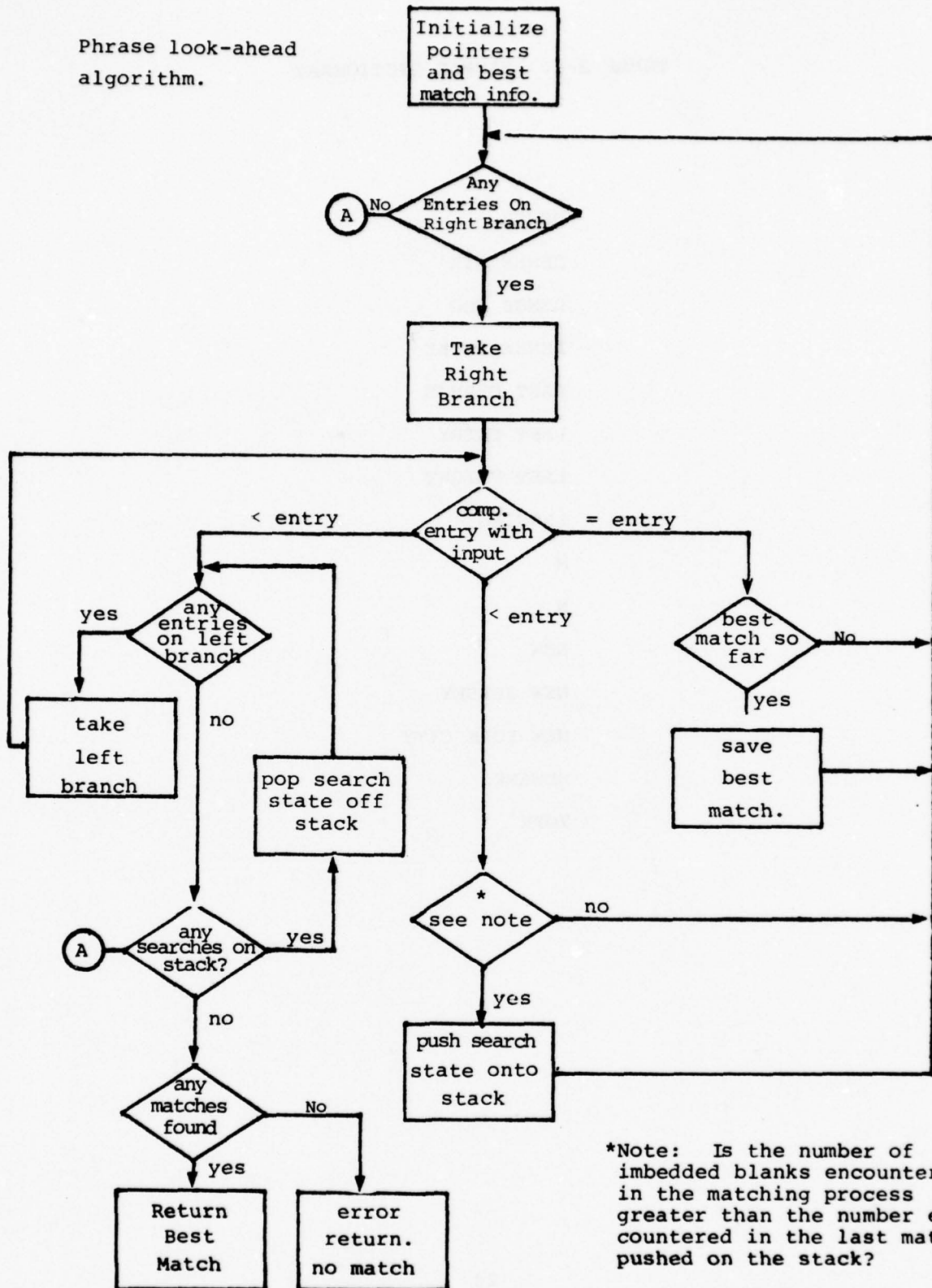
NEW JERSEY

NEW YORK CITY

NEWARK

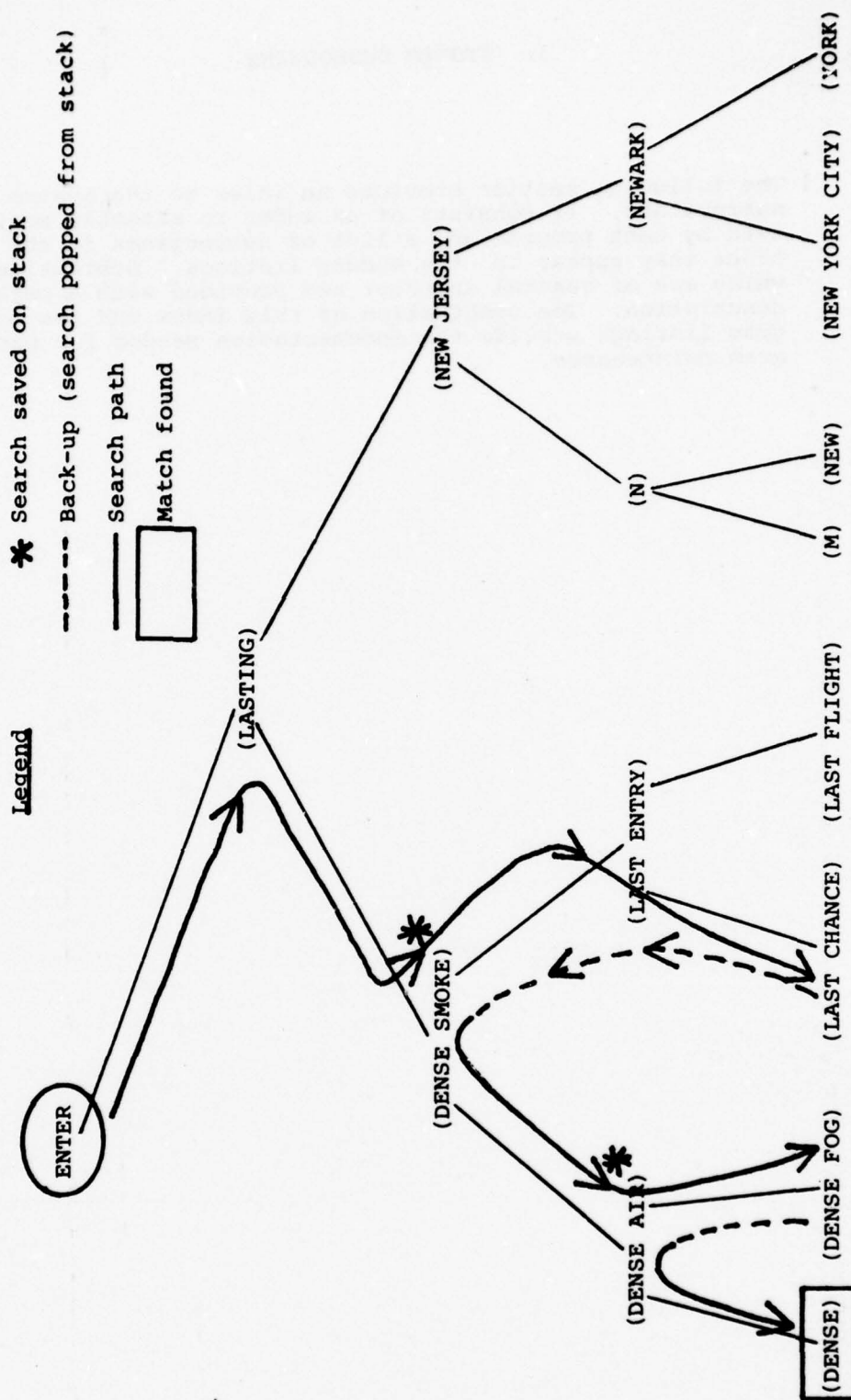
YORK

Phrase look-ahead  
algorithm.



\*Note: Is the number of imbedded blanks encountered in the matching process greater than the number encountered in the last match pushed on the stack?

Path followed through dictionary to find best match for input string of "DENSE TRAFFIC"



### 3. SYSTEM SUBROUTINE

The following section provides an index to the system subroutines. It consists of an index to assembly modules used by each program and a list of subroutines in the order they appear in the module listings. Subroutines which are of general interest are provided with a working description. The combination of this index and the program listings provide the documentation needed for program maintenance.



### 3.1 PROGRAM ASSEMBLY MODULE NAMES

Program	VEDIT
Assembly modules	VEDCSI
	STKBUF
	COMBLK
	CTAB
	DIRPAG
	ERRORS
	LIST
	TALK
	INSERT
	DELETE
	GARBG
Program	RECORD
Assembly modules	RECCSI
	STKBUF
	LISTEN
	ADPCM
	CTAB
	GLOBAL
Program	H516
Assembly modules	H516
	STKBUF
	LOK516
	PLY516
	CTAB
	SPGLBL
Other modules:	PARAMS
	Contains assembly parameters

### 3.2 SUBROUTINE NAMES

Module	VEDCSI
Subroutines	SYSINT - system initialize
	CSI - main program loop
	GCOM - get command
	CLKUP - command table lookup
	CSMTCH - command string match
	SWTCHK - legal switch checking routine
	FLBLK - flush leading break characters
	BRKCH - check for break character
	TTYOT - console output
	CRLF - print carriage return line feed
	OCTIN - octal number input
	OCTOUT - octal number output
	LISTNM - list a dictionary name
	CNFRM - confirm a request for an operation
	AYS - "Are you sure?" command verify
	FCE - fatal consistency error
Module	STKBUF
Subroutines	PUSH - stack push
	RTRN - normal subroutine return
	ERTRN - subroutine error return
	BUFINT - ring buffer initialize
	RIN - ring buffer in
	LINSET - ring buffer limit set
	RINC - input with limit set
	RBKUP - input pointer backup
	CROUT - conditional output
	BUFRST - reset conditional output
	BUFSET - output set
	CRBKUP - conditional backup
	ROUT - ring buffer output

Module	COMBLK
Subroutines	KILL - dictionary reinitialize BYE - program exit Also contains command table used for command lookup.
Module:	CTAB contains all tables used by both ADPCM encode and decode.
Module	DIRPAG
Subroutines	VMINT - "Virtual memory" initialize RDBYT WRBYT - word and byte i/o on dictionary RDWRD via "virtual memory" WRWRD VMNG - virtual memory manager VMBKUP - back up virtual memory or disk DIRINT - directory initialize RDDE WRDE - directory entry read and write RDFSE WRFSE - free storage entry read and write NFSPTR - get next entry with same file pointer CRFSE - create free storage entry FSPACK - pack contiguous free storage DELFSE - delete free storage entry GUID - generate unique identifier for file CRDIR - create a dictionary entry CRDCT - create a text name for a dictionary entry DELDE - delete dictionary entry DCTBM - get best match in dictionary DLKUP - binary search dictionary lookup DMTCH - match single dictionary entry STRMTC - as above best for wild card option

Module	DIRPAG (Continued)
Subroutines	FLMTCH - match entry with full command string ARGINT - initialize NXTARG routine NXTARG - return successive entries in dictionary which match command string Also program global variables.
Module	ERRORS
Subroutines	PRERR - print error message PRCT - print current token in command Also error messages (ASCII TEXT)
Module	LIST
Subroutines	LIST - program list command LSTCHK - check arg to see if it should be listed LSTHDR - print listing header LSTPRT - print file name LSTFS - list file area free storage
Module	TALK
Subroutines	TALK - program talk command SPINT - speaking buffer initialize PBFINT - buffer set up FILBUF - maintain speaking buffers ADSTRT - start of A/D convertor (used as clock) PBINT - interrupt service and ADPCM decode



Module	DELETE
Subroutines	DELETE - program DELETE command
Module	INSERT
Subroutines	INSERT SYNON RENAME - program commands ENTER
Module	GARBG
Subroutines	GARBG - program command for free storage "garbage collections"
Module	RECCSI
Subroutines	CSI - program main loop GCOM - get command CLKUP - command lookup CSMTCH - command string match FLBLK - flush blanks BRKCH - check for break characters TTYOT - console output CRLF - print carriage return line feed PRERR - print error message PRCT - print current token AYS - "Are you sure?" FCE - fatal consistency error
Module	GLOBAL Contains all program global variables
Module	LISTEN
Subroutines	LISTEN - program LISTEN command PRGINT - initialize BFRSTP - buffer set up DSKRCD - disk recording routine ADINT - program interrupt handler PRGFIN - program close GAIN - program command to set GAIN on A/D

Module	ADPCM
Subroutines	ADPST - initialize SAVE - program - main loop WDPACK - code word packing DOPEN - dictionary open command FRALOC - free storage allocation WRINT disk word at a time i/o RDINT initialize WRCLS - close write channel WRTWAT REDWAT word at a time disk i/o AUTORD ADPCM - ADPCM word encode WBINS - insert sample in energy "window" FBSQ - compute $(C(i)-7.5)^2$ PERMFL - make a permanent speech file DCLOSE - close dictionary DELFSE - delete free storage entry NSFPTR - get next file with same file pointer THRESH - program threshold modification

Module	H516
Subroutine	H516 - program main loop GLIN - input time of ASCII text CPYLUUD - copy in dictionary for lookup CPYPBD - copy in dictionary for speaking ALCCOR - allocate free core TTYLOT - console output CRLF - carriage return - line feed PRERR - print error PRCT - print current token BRKCH - check for breaks
Module	LOK516
Subroutines	PRSLIN - look up text in a single line PAUSE - pause handler for punctuation AUTOWR - write pointer into ring buffer NUMCHK - check if text string is a numeral SINGLE - check if token is a single character PRSNUM - parse up a numeral PRS3DG - parse up 3 digits of a numeral PRSWRD - spell a word PBLKUP - phrase look ahead binary search PBMTCH - match single dictionary entry WRINT RDINT - initialize word at a time i/o WRCLS - close write channel WRWAT - write disk word at a time RDWAT - read disk word at a time AUTORD - auto read of words

Module	PLY516
Subroutines	PLAYBK - main loop
	SPINT - speaking initialize
	PBFINT - buffer initialize
	FILBUF - keep buffers full
	ADSTRT - start A/D converter (clock)
	PBINT - interrupt handler and ADPCM decode
	also variables and tables

Module	SPGLBL
Subroutines	Speak program global variables



### 3.3 SUBROUTINE DESCRIPTION

Macro: .CAL

Function: .CALL subroutine. Provides uniform format for calling subroutines which have error return.

Other Operations: .CAL sub, error.

Expands into:

JSR R7, sub  
error

"sub" is the name of the subroutine to be called.

"error" is the address of the error handler concerned with an error return from the subroutine.

.....

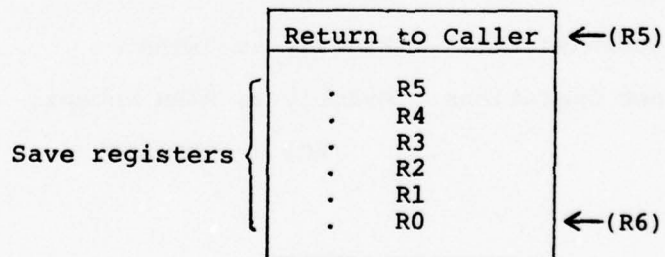
Name: PUSH

Function: PUSH all registers on stack, also position R5.

Called: JSR R5, PUSH

Arguments: None.

Other Operations: Upon return from PUSH, user stack looks like this:



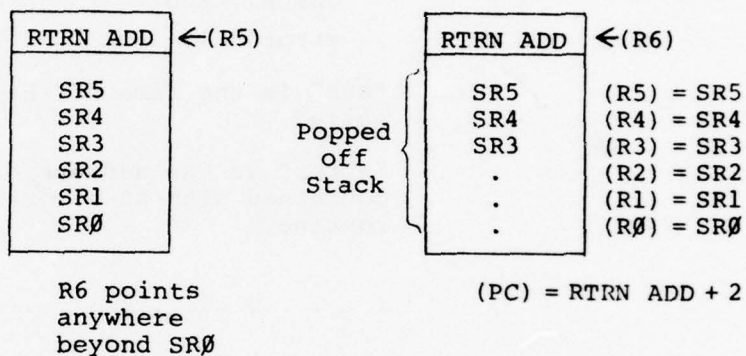
Name: RTRN

Function: RETURN from subroutine; restores registers from stack and makes a return, skipping error return location in calling routine.

Called: JMP RTRN

Arguments: R6 need not point to stack.  
R5, however, must point as shown in "PUSH" description.

Other Operations: Stack before call                      after



Name: ERTRN

Function: Error ReTURN. Return from subroutine after restoring registers. Return made by indirecting through error address pointed to by return address in stack.

Called: JMP ERTRN

Arguments: Exactly as "RTRN".

Other Operations: Exactly as RTRN except:  
(PC) = (RTRNADD)

## RING BUFFER PACKAGE

The ring buffer package for the VRS is a versatile system of routines for the buffering of data between two programs. Features include those of a normal ring buffer as well as the ability to examine the contents of the buffer without actually removing the item from the buffer, as would be desirable for comparisons at an input stream with many text strings. An additional feature is a limit pointer useful for line-at-a-time editing. The limit pointer can be set to the point where an input line ends. While this line is being processed, a new line is composed. The buffer can only be processed up to the limit pointer, even though the input pointer is beyond that point. This prevents processing on a partially composed line.

Consulting Figure 3-1, the function of each of the pointers is as follows:

- IN - this is the input pointer to the buffer. As data is added, the pointer moves toward out (clockwise). Invalid data can be removed by "backing up" (counter clockwise). The limit on the clockwise direction, is the OUT pointer. At that point the buffer is full. The back up limit is the LIN pointer.
- LIN - this is the input and output limit pointer. It is usually pointed to the last new line character in the input stream. Once set, the input pointer cannot be backed up beyond it, nor can the output pointer move forward past it; that is, the output pointer cannot remove data beyond the limit pointer. LIN can only move clockwise. It is set to the position of IN by a subroutine call, and remains stationary until the next call regardless of the motion of IN or OUT.
- OUT - the ring buffer output pointer. Removal of data is "final" in the sense that this pointer cannot be backed up. It sets the limit up to which IN can insert data. It is either moved one character at a time, or moved by a subroutine call to the position of COUT.
- COUT - this pointer allows examination of any data between the LIN and the OUT pointers in the area shown on Figure 3-1. It can move one character at a time in either direction or be set to the value at OUT.

Ring Buffer Package.

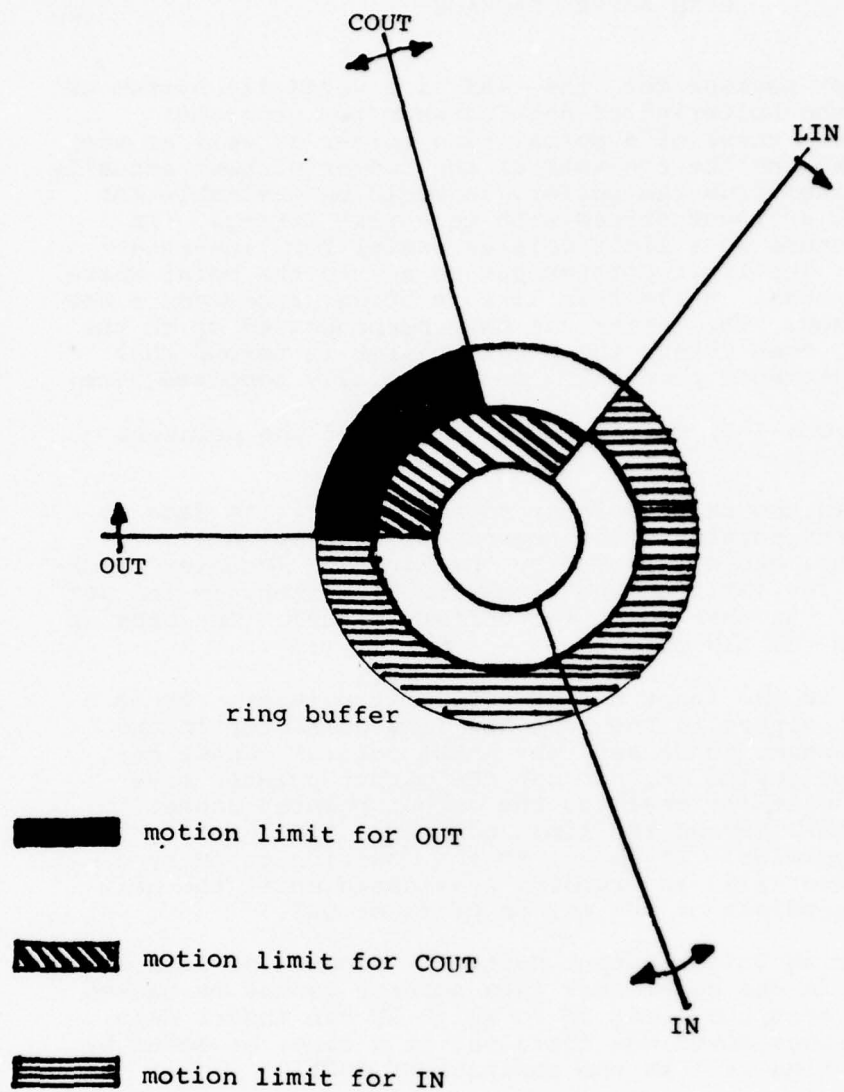


FIGURE 3-1. POINTER SUMMARY ARROWS INDICATE DIRECTION OF MOTION.



Name: BUFINT  
 Functions: BUffer INItialize. Set up a ring buffer in core.  
 Called: .CAL BUFINT, error.  
 Inputs: RØ pointer to 262<sub>10</sub> byte block of storage to be used as a ring buffer.  
 Outputs: None.  
 Other Operations: 256<sub>10</sub> byte circular buffer is initialized in core. 6 byte header is provided as follows:

<u>Byte</u>	<u>Name</u>	<u>Function</u>	<u>Initial Value</u>
Ø	LIN	Line Limit. Sets limit beyond which output pointers cannot go.	Ø
1	IN	Input pointer. Points to next free byte in buffer.	Ø
2	CFUL	Conditional fullness. Distance from conditional output pointer to output pointer.	Ø
3	COUT	Conditional output pointer. Free-flowing pointer to examine any character between LIN and OUT without removing from buffer.	Ø
4	FUL	Actual fullness. Distance from output pointer to input.	Ø

<u>Byte</u>	<u>Name</u>	<u>Function</u>	<u>Initial Value</u>
5	OUT	Output pointer. Final output pointer for characters.	Ø

Routines called: None.

Errors: None.

.....

Name: RIN

Function: Ring buffer IInput. Places input character  
in ring buffer.

Called: .CAL RIN, error.

# Args: Two.

Inputs: RØ - pointer to ring buffer  
R1 - character to be inserted right justified.

Outputs: None.

Other Operations: Fullness count and input pointer incremented after character inserted. (Also conditional fullness updated.)

Routines called: None.

Errors: Buffer full.

Name: LINSET  
 Function: LIN pointer SET. Limit pointer set to current value at input pointer.  
 Called: .CAL LINSET, error.  
 Inputs: RØ - pointer to ring buffer.  
 Outputs: None.  
 Other Operations: Contents of IN pointer placed in LIN pointer.  
 Routines Called: None.  
 Errors: None.

. . . . .

Name: RINC  
 Function: Ring buffer Inter with No Check. Equivalent to RIN followed by LINSET. Used when data need not be checked before definitely entering it.  
 Called: .CAL RINC, error.  
 Inputs: RØ - pointer to ring buffer.  
 R1 - character to be inserted.  
 Outputs: None.  
 Results: If error - none.  
 If no error, character placed in buffer, conditional fullness and fullness updated, limit pointer (LIN) and input pointer both set to next character.  
 Routines Called: None.  
 Errors: Buffer already full.

Name: RBKUP  
 Function: Ring buffer BackUP. Removes last character placed in ring buffer.  
 Called: .CAL RBKUP, error.  
 # Args: Two.  
 Inputs: RØ - pointer to ring buffer used.  
 Outputs: SR1 - if no error, returns character removed - if error, returns unchanged.  
 Other Operations: Pointer positions moved.  
 Routines Called: None.  
 Errors: Pointer already backed up to LIN limit pointer.

.....

Name: CROUT  
 Function: Conditional Ring buffer OUTput. Remove next character from buffer using the conditional ring buffer pointer.  
 Called: .CAL CROUT, error.  
 # Args: Two.  
 Inputs: RØ - pointer to ring buffer.  
 Outputs: SR1 - character removed from buffer. If error, no change.  
 Other Operations: Conditional pointer incremented, conditional fullness decremented if no error, otherwise no change.  
 Routines Called: None.  
 Errors: Buffer "empty".  
 Conditional output pointer COUT has caught up to limit pointer LIN.

Name: BUFRST

Function: BUffer ReSeT. Resets conditional output and conditional fullness to value of output and fullness.

Called: .CAL BUFRST, error.

# Args: None.

Other Operations: Covered in function.

Errors: None.

.....

Name: BUFSET

Function: BUffer SET. Moves output and fullness up to conditional output and conditional fullness.

Called: .CAL BUFSET, error.

# Args: None.

Other Operations: Covered in function.

Errors: None.



Name: CRBKUP  
 Function: Condition Ring Buffer output BackUP.  
 Back up to previous character.  
 Called: .CAL RBKUP, error.  
 # Args: Two.  
 Inputs: R0 - pointer to ring buffer.  
 Outputs: SR1 - character removed from buffer.  
 If error, SR1 is unchanged.  
 Routines Called: None.  
 Errors: Attempt to back up beyond output pointer.  
 Other Operations: Conditional fullness incremented if no  
 error. COUT backed up one character.  
 . . . . .  
 Name: ROUT  
 Function: Ring buffer OUTput. Removes next character  
 pointed to by output pointer.  
 Called: .CAL ROUT, error.  
 # Args. Two.  
 Inputs: R0 - pointer to ring buffer.  
 Outputs: R1 - no error: Character removed.  
 R1 - error: Unchanged.  
 Other Operations: Error - none.  
 No error - output incremented, fullness  
 decremented.  
 Routines Called: None.  
 Errors: Attempt to move output pointer past con-  
 ditional output pointer.

Name: VMINT

Function: Virtual Memory INiTialize. Sets up core buffers for disk to permit read and write of disk resident dictionary through a virtual memory system.

Called: .CAL VMINT, error.

Arguments: None.

Other Operations: Core Buffers initialized to contain core keys of first five pages (256 words each) of the dictionary. Core buffers brought in by an LRU algorithm, so appropriate variables for LRU are initialized.

Routines Called: None.  
RT-11 monitor calls .READW

Errors: RT-11 errors only - disk read.  
Error message pointer returned in ERPNTR.

Name: RDBYT - read one byte  
RDWRD - read one word  
WRBYT - write one byte  
WRWRD - write one word

Function: Basic input-output for dictionary.

Called: .CAL name, error.

Arguments: R2 - points to word or byte on disk, if  
it is a word, it should point to an even  
byte boundary.  
  
R1 - argument to be read or written.  
Byte arguments should be right justified.  
On return from read byte, top byte cleared.

Other Operations: LRU variables updated. If desired, data  
not in core at time of call is swapped in.

Routines called: VMNG

Errors: RT-11 errors.  
Error message pointer returned in ERPNTR.

Name: VMBKUP

Function: Virtual Memory Back UP. Updates disk resident copy of dictionary by swapping out pages in core buffers.

Called: .CAL VMBKUP, error.

Arguments: None.

Other Operations: None other than described in function.

Routines Called: RT-11 monitor calls .WRITW

Errors: RT-11 errors.  
Error message pointer returned.

Name: DIRINT  
Function: DIrectory INitialize.  
Called: .CAL DIRINT, error.  
Arguments: None.  
Other Operations: All variables in header at dictionary set  
to indicate empty dictionary.  
Routines Called: CRFSE.  
Errors: Error return from CRFSE returned directly.



Name: RDFSE - Read free storage entry.  
WRFSE - Write free storage entry.

Function: i/o on 2-word free storage information.

Called: .CAL Name, error.

Arguments: R2 - byte pointer to first byte of two-word descriptor.  
Callers stack at call time:  
    (R6) - address of block of free storage.  
    2(R6) - size of free storage in number of blocks.

Other Operations: Dictionary written via WRWRD.

Routines called: RDWRD  
WRWRD

Errors: Returned directly from above routines.

Name: RDDE - read directory entry.  
WRDE - write directory entry.

Function: Read or write a three-word file information block from dictionary.

Called: .CAL name, error.

Arguments: R2 - pointer to first byte at entry to be accessed. (Offset from beginning of dictionary.) Callers stack as follows (at call time):  
(R6) - pointer to text name of file.  
2(R6) - file size information.  
4(R6) - pointer to first block of file.

Other Operations: On WRDE, dictionary written through virtual memory.

Routines called: WRWRD  
RDWRD

Errors: Returned directly from above calls.

Name: CRFSE

Function: Create a two-word entry in the table of free storage space.

Called: .CAL CRFSE, error.

Arguments: Stack as follows:  
    (R6) - address of disk area to be returned to free storage.  
    2(R6) - size in blocks of disk area.

Other Operations: Free storage area sorted by address. Entries moved to accommodate new entry. If new entry is contiguous with existing entry, the entries are merged into one entry. Pointers to table of free storage space are updated.

Routines Called: RDFSE  
                  WRFSE  
                  FSPACK

Errors: Free storage space full. Error pointer returned in ERPNTR.  
  
Other errors directly returned from called routines.

Name: DELFSE  
 Function: Delete FREE storage table entry.  
 Called: .CAL DELFSE, error.  
 Arguments: R2 pointer to entry to be removed.  
 Other Operations: Table of free storage updated by moving  
 remaining entries down over deleted entry  
 and by updating pointers to table.  
 Routines Called: RDFSE  
 WRFSE  
 Errors: Returned directly from routines called.

.....

Name: GUID  
 Function: Generate Unique IDentifier used in file  
 Creation.  
 Called: .CAL GUID, error.  
 Arguments: R3 - high order byte of unique identifier  
 returned in low order byte of R3. High  
 order byte of R3 cleared.  
 R4 - low order two bytes of uid.  
 Other Operations: Current uid in file header updated.  
 Errors: None.



**Name:** CRDIR

**Function:** CReate Dictionary entry. Takes text argument and creates a new dictionary entry. File block size is initially zero. Synonyms use three byte unique identifier copied into last block size, and file pointer fields of entry.

**Called:** .CAL CRDIR, error.

**Arguments:** R0 - pointer ring buffer containing text name for new entry. First call to CROUT should return first character for name. Name used until first non-blank break character.

R2 - insertion point for new entry.

**Other Operations:** Text entry inserted after previous end of text area. (File name area.) New three-word entry inserted in dictionary. Pointers to dictionary updated.

**Routines Called:** CRDCT

RDDE

WRDE

GUID

**Errors:** Dictionary full (returned in ERPNTR) or else error returned from routines called.



Name: CRDCT

Function: Inserts text name into file name area of dictionary. Returns pointer to newly created name.

Called: .CAL CRDCT, error.

Arguments: R0 - pointer to ring buffer containing text name described in CRDIR.  
R1 - returns pointer to location. Text name was inserted.

Other Operations: File name area and appropriate pointer to that area are updated.

Routines Called: CROUT  
BRKCH  
WRBYT

Errors: Dictionary full - returned in ERPNTR.  
Or else error returned from routines called.

**Name:** NFSPTR

**Function:** Find next entry with same file pointer. Used to locate synonyms to a file. When called, look for synonym which alphabetically follows after entry provided as argument. If search runs past end of dictionary, restart at the beginning of the dictionary. If an entry has no synonyms, return original entry.

**Called:** .CAL NSFPTR, error.

**Arguments:** On entry R2 is pointer to three-word entry block in dictionary; on exit R2 contains pointer to three-word block of next synonym.

**Other Operations:** None.

**Routines Called:** RDDE.

**Errors:** Returned directly from RDDE.

Name: DELDE

Function: Delete dictionary entry - remove three-word block associated with entry and the text entry name.

Called: .CAL DELDE, error.

Arguments: R2 - pointer to three-word block of dictionary entry to be removed.

Other Operations: Text entry removed from file name area. Any names in higher core than removed name are moved down to compress file name area. The same is done for the information block area. All pointers are updated. LCHECK is also updated if entry it points to is moved by delete.

Routines Called: RDDE  
RDBYT  
WRBYT  
RDWRD  
WRWRD

Errors: Returned directly from routines called.

**Name:** DCTBM

**Function:** Find best match in dictionary. Determines if string provided provides a match with an entry in the dictionary up until the first non-blank break character.

**Called:** .CAL DCTBM, error.

**Arguments:** R0 - points to ring buffer with string in it. If match output pointer points to beginning of string with leading blanks flushed, conditional output pointer points to break character terminating string. If no match, both output pointers point to beginning of string with leading blanks flushed.

R2 - pointer to match. If match occurs, R2 is pointed to entry in dictionary which matches the string. If no match, R2 points to place in dictionary where new entry would be inserted if input string were used for test name. If end of text encountered in input string before any other text encountered, R2 is set to all ones.

**Other Operations:** None.

**Routines Called:** FLBLK  
DLKUP  
CRBKUP  
CRDUT  
BRKCH  
BUFRST

**Errors:** Text string does not match any entries or end of text in input string.

ERPNTNTR unaffected by DCTBM. If set by routines called, its value is returned unchanged.



Name: DLKUP

Function: Dictionary LookUP. Performs a binary search on dictionary to find entry which matches input string.

Called: .CAL DLKUP, error.

Arguments: R0 - points to text string in ring buffer. Both output pointers must point to the first character of the string. If no match is found, output pointers are unchanged. If match occurs, conditional output pointer points to break character at end of match.

R2 - if match occurs in course of search, R2 points to matching entry. If no match, R2 points to insertion point found for string. If end of text, R2 is set to all ones.

R3 - if match found, R3 points to insertion point for string. This is done because a string which matches an entry may continue beyond the match. For example, "NEW YORK CITY" would match "NEW YORK" in the dictionary but could still be inserted as a new entry.

Other Operations: None.

Routines Called: BUFRST

DMTCH

Errors: No match in dictionary or end of text in ring buffer.



#### 4. DATA BASE

The File System for the VRS was designed to meet three criteria:

- a. Speed - The File System must be capable of a data rate of 7.5 disk reads per second.
- b. Compact Dictionary - The File System eventually must maintain a full dictionary of 4000 entries. To permit file lookup to proceed at maximum speed the dictionary must be core resident. The amount of information associated with each dictionary entry must be minimized in order to keep a 4000 entry dictionary to manageable size.
- c. Editing Flexibility - The dictionary must be easily modifiable by the system programs. The user should not be subjected to constraints due to limitations in the file structure.

In addition to the above criteria, the dictionary format must permit phrase look-ahead.

In general, these criteria conflict. Speed and editing flexibility always require additional information, which implies additional space. The design chosen provides maximum speed in operation but permits the desired editing capabilities through software which calculates the required pointers, rather than storing them in the dictionary.

##### File System Description

The VRS file system is divided into five major storage areas. These areas (illustrated in Figure 1-2) are:

- a. Header Block - This area contains a 348 byte descriptor for the file system. This includes an 8 byte name block and information concerning the size of the remaining 4 areas.
- b. File Name Area - The text names for the various dictionary entries are stored in this area. As the names can be of any length, no fixed size is set for an entry. Instead, each name is followed by a zero byte.
- c. File Description Area - The size and location of each dictionary entry is stored in this area.

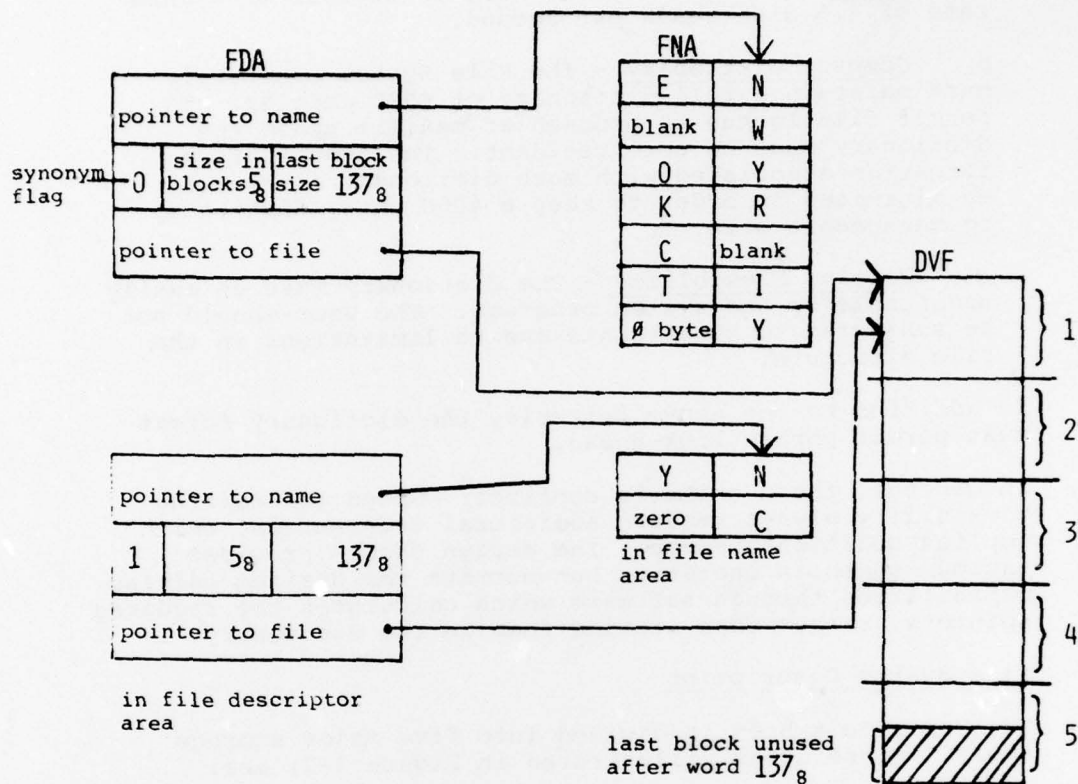


FIGURE 4-1. SAMPLE FILE STRUCTURE CREATED FOR THE FILE NAME "NEW YORK CITY" AND ITS ABBREVIATION (synonym) "NYC"

d. Free Storage Description Area - Unused disk storage is described in this area.

e. Digitized Voice File Area - Contains the encoded speech.

These areas are all on disk in the layout shown in Figure 4-2. The descriptor areas are copied into core memory as needed by the particular program using the file system. (For example see flowchart for "SPEAK" in Section 2.)

To visualize the functioning of these various areas it is best to examine a typical dictionary entry (Figure 1-3). The entry shown is an example of how the encoded utterance "New York City" might be accessed by that name as well as by the abbreviation "NYC". The files are accessed first through the file descriptor area. This area consists of a number of three-word blocks which appear in detail in the example. Most of the important capabilities are provided by the information in these blocks.

The first word points to the file name, the second contains the file status and size information, and the third points to the disk location. Alphabetic sorting is accomplished by moving these descriptor blocks, rather than directly moving the variable-length entries in the file name area.

The file name can be of arbitrary length, contain embedded blanks, and must be terminated by a zero byte. New names are added to the end of the name area, while the corresponding three-word block is inserted in the correct position in the block area as determined by the sort.

The file status and size (second word of descriptor block) contains the following information: The file size (in 256 word blocks) is contained in the low order 7 bits of the high order byte. The number of words actually used in the last block is contained in the low order byte. The high order bit of the word is used for abbreviations. When a dictionary entry is first created, the high order bit, called a "synonym flag" is left zero. If an entry is being created as an abbreviation for an already existing entry, the remainder of the second and third words are copies from the original entry. During editing, an entry's synonyms are found by comparing the file pointer and size words for a match with other dictionary entries. Note that an abbreviation entry is identical in every respect to other entries except for the flag. When an empty file is created, a three byte "unique identifier" maintained in the file header is copied into the file pointer and size of the last block portions of the dictionary entry. This provides a means of distinguishing synonyms before a file is created. When a file is deleted, the block size and the file pointer are copied into the free storage area to give information about available space in the file storage area.

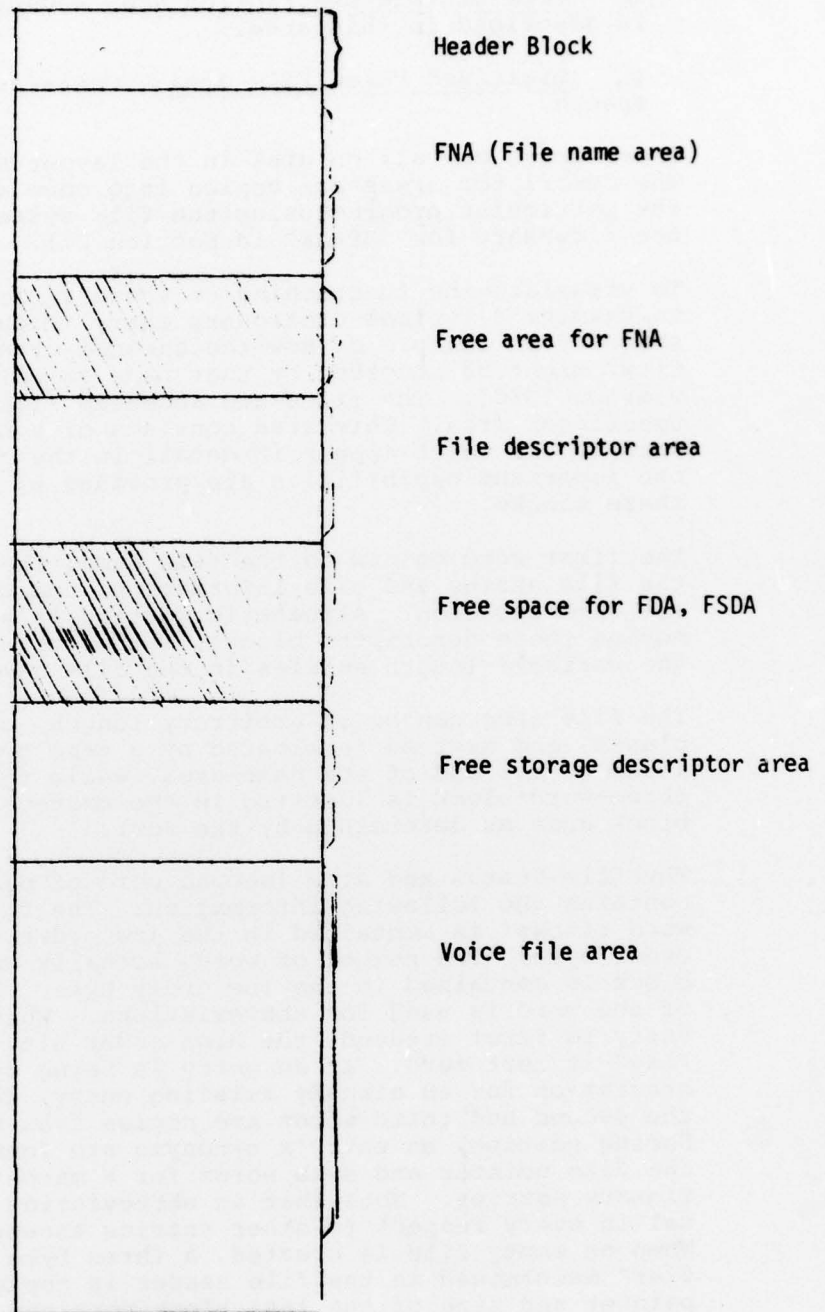


FIGURE 4-2 - FILE SYSTEM PARTITION